SMT Solvers – A Promising Way for Verifying User Interfaces?

Jens Bruchertseifer
jens.bruchertseifer@uni-trier.de
University of Trier
Trier, Germany

Benjamin Weyers
weyers@uni-trier.de
University of Trier
Trier, Germany

ABSTRACT

Evaluating user interfaces of interactive applications in terms of correctness has always been a lengthy and error-prone task, as it would be executed by humans. This has led to this aspect of development being severely neglected, which poses a fundamental problem in terms of usability. In this work, we present a novel approach based on the creation of a formal representation from a given user interface, which is transformed into a reference net and finally converted into an SMT formula. As SMT formulae can be evaluated easily using highly efficient SMT solvers, this approach promises good efficiency in real-world scenarios.

KEYWORDS

formal methods in HCI, SMT, SMT solver, Z3, UI model, verification, reference nets, Renew, FILL, transformation, automatization

1 INTRODUCTION

Since the middle of the past century, user interfaces have developed from a charming idea into a serious professional tool indispensable for digital interactive systems [20]. They can be found nearly everywhere, even in areas where errors can cause tremendous financial and human costs [18, 26]. This evolution was also accompanied by the emergence of more complex interfaces embodying highly complex interaction processes, specifically in case of safety critical systems such as aircrafts [2] or medical devices [14]. Therefore, this fundamental shift in their relevance and size increased the relevance of ensuring their validity in regards of usability criteria. This includes that users can perform their tasks effectively with a given user interface. Hence, in this paper we will focus on a fundamental aspect of effectiveness as usability criteria, which we will depict as correctness of a user interface. In this work, we understand correctness as whether the actual functionality offered by a user interface to a user fits previously defined requirements in terms of its effectiveness. Thus, if correctness cannot be ensured, a user will not be able to work with a user interface in a meaningful way, regardless of how it is designed. Two processes of showing correctness are often distinguished: validation and verification as defined below, where validation focuses on showing whether a user interface is doing the right thing, whereas verification rather aims at showing that a user interface is doing the thing right (following the terminology as presented in [11]).

Various approaches have been suggested to validate or verify user interfaces in terms of various types of requirements. For instance, model checking approaches have been used to investigate the actual interaction space of given user interfaces to check whether unwanted states of the system can be reached by interacting with the given user interface [7, 8]. Other examples are using formal specification and simulation to show the correctness of user interfaces such as presented by Bowen et al. [5, 6]. A further approach has been presented, which uses Petri net-based models for describing the functional space of a user interface and thus being able to check for dead locks or other relevant characteristics of the created model [3, 17].

Since the underlying problem is a computationally very complex one, this is a costly, error-prone and lengthy task, so that modern user interfaces are usually only verified in excerpts, specifically in cases where safety concerns apply [2, 14, 15].

Therefore, there is still a need for further research in terms of verification approaches as only small examples of interactive systems can be verified due to high computational costs or only experts are able to apply verification methods to a given user interface.

Therefore, this work will present an outline of an automated system for verifying user interfaces. The automation lies in

(a) using transformation algorithms based on formal description concepts,
(b) a visual modeling language aiming at reducing the needed expertise to model and describe interaction logic and
(c) a verification method, which picks up standardized methods and implementations and thus relies on well developed approaches.

In this regard, methods are desirable that already operate close to the maximum possible efficiency. Checking the satisfiability of propositional logic formulae (SAT) has been used to model complex situations since the early days of computer science [9] and automated solvers are extremely efficient nowadays [12, 21], which makes it a reasonable approach for our work. Thus, we will concentrate on SMT (satisfiability modulo theories) formulae instead of the classic SAT formulae, since the former offer the concept of data types in the solving process, which simplifies the comprehensibility, but are still about as efficiently solvable as the latter [16]. A reference implementation is currently work in progress and will support this proposition soon, giving the possibility to evaluate code of an existing software in order to derive and verify a formal model of it. This paper at hand aims at describing the overall process as well as the central idea of using SMT solvers in terms of the verification of user interfaces with a specific focus on effectiveness as usability criteria. This novel approach should support the evaluation process of user interfaces in terms of correctness fundamentally and also

Veröffentlicht durch die Gesellschaft für Informatik e.V.
in K. Marky, U. Grünefeld & T. Kosch (Hrsg.):
Mensch und Computer 2022 – Workshopband, 04.-07. September 2022, Darmstadt
© 2022 Copyright held by the owner/author(s).
https://doi.org/10.18420/muc2022-mci-ws06-450

enable users to verify adaptations applied to a given user interface on the go (i.e. [24]).

2 METHOD

The method presented in this paper consists of a multi-tier approach. It is based on a model analogously to the one used in [23] based on three layers, which consisted of the physical representation, the interaction logic and the system layer. Those three layers were defined as follows:

(1) The physical representation contained all graphical widgets and all elements the user can use for immediate interaction; for example buttons, checkboxes or tooltips.
(2) Below this – seen from a logical perspective – there exists the interaction logic defining the behavior of all elements which can be contained in the physical representation, be it either a reaction to events on the physical elements or a reaction to information from the system, which represents the bottom layer.
(3) Since viewing the system in full detail would increase the difficulty of a correct separation of the layers and complicate a clear isolation of user interface issues, the bottom layer is assumed as a black box, but is characterized by a well-defined state set.

This architecture poses some similarities to slightly extended versions of the famous Model-View-Controller pattern.

For modeling the actual behavior of a user interface, in this case the interaction logic, we used a visual language called Formal Interaction Logic Language (FILL). It should be noted that FILL has roots in the Business Process Modelling Notation (BPMN)[1] maintained by the Object Management Group (OMG) to some extent in terms of basic formalisms. For a solid introduction into BPMN, we refer to works like [1]. The major goal of this work at hand is that the interaction logic is taken and processed into an SMT formula, which requires multiple complex steps, as explained below. Afterwards, this formula can be combined with SMT formulae representing conditions to test against, such that an SMT solver is able to check this combined formula for correctness. If the requirements are included in negated form into the formulae and the SMT solver is able to find a model for it, we can infer that the interaction logic violates the actual condition or requirement.

As this approach is not trivial, a short graphical overview of this idea is presented in Figure 1.

2.1 Interaction Logic – The Gist of the Matter

As mentioned in [22], interaction can be basically described as data exchanges while conforming to a certain formalism, whereas interaction logic can be defined as follows:

Definition 2.1. Interaction logic is a model of interaction representing a set of models of interactions processes. *Interaction processes* model processes of data flow between a system interface and its environment. Interaction logic can be described formally, semi-formally or informally.

It should be clear that this work will focus on the formal description of interaction processes. Therefore, we will make use

[1] https://www.omg.org/spec/BPMN/2.0/

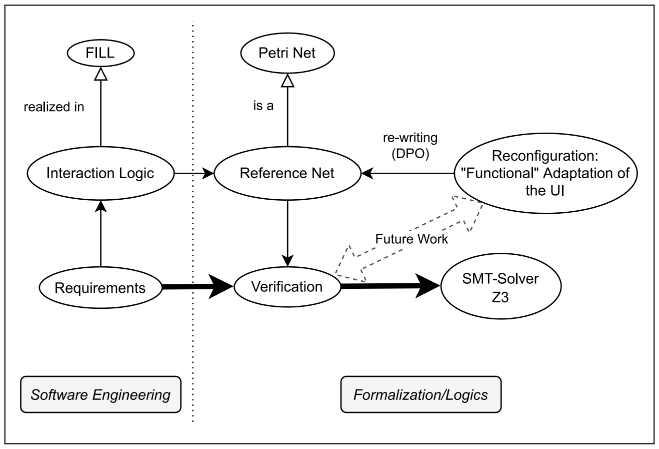

Figure 1: A structural overview of the concept: The figure uses a notation slightly similar to UML. Thin arrows with filled head denote data flows, while ones with empty arrow head specify additional information. Bold arrows denote the functionalities described in this work. The dashed bold arrow denotes future work.

of FILL, which was introduced in [22]. FILL is a modelling language describing the processing of user events or data between the interface a user is presented and the system to be controlled, which is assumed as a black box, in accordance with the previous remarks. In a nutshell, FILL models the interaction logic as system and interaction-logic operations together with input and output ports (connecting an operation to a different one or a proxy), as well as input and output proxies (connecting to elements from the physical layer) and at last two sets of tuples characterizing the non-static part of the interaction logic.

More precisely, we will use an extended variant of FILL called $\text{TFILL}_{\text{EXT}}$. This variant is additionally capable of representing complex data types, which are assigned to every port and proxy, as well as – in short – capable of modelling more complex interaction processes by supporting more elements like input and output channels or even elements from the language BPMN. This is very helpful for creating branches or combinations in interaction processes.

For FILL, we aim to use an implementation in Java, which has the advantage of being cross-platform usable. This also offers the option to use Java annotations for the definition of invariants as well as the syntax.

2.2 Interaction Logic as Reference Nets

In order to generate automatically evaluable and executable representations, those FILL expressions are converted to reference nets. As a variant of the famous Petri nets, the latter introduced by Carl A. Petri in [19], reference nets provide on the one hand – a quite well-understood – semantics. Reference nets were introduced in [13] and provide better options for representing interaction logic as well as a more high-level view, since they support complex data types for example. They are very suitable for this use-case, since they are also a deterministic graph-based representation and offer a way to describe (typed) data objects, which is of interest, because this approach should not be realized entirely on the basic Boolean level, as mentioned initially. Another aspect supporting the usage of reference nets is the availability of well-supported tools, like Renew[2].

These properties have also gained attention in other works on formal modelling of user interfaces [25]. In the present work, only the necessary aspects of Petri nets and reference nets are covered, since more is not within the scope of the present work. For a detailed general introduction, we refer to standard textbooks like [4].

The conversion between FILL-based representation and reference nets was already presented in [22]. As this process is non-trivial and represents not the focus of this work, we refer for a detailed explanation to the original work. In short, this procedure takes an empty reference net and the given (non-empty) FILL instance and transforms it element by element. It can be noted that the conversion does not increase the graph size by much, since most of the operations can be modeled in a similar way again, e.g. system and interaction-logic operations are transformed into places (1:1) and operations are transformed into two transitions (call/return value).

In this case, we also aim for the usage of Java, since it is also part of the reference net's inscription language, and was also used for the the systems presented in [13].

2.3 The Gap – From Reference Nets to SMT Formulae

These reference net-based representations of the interaction logic are now transformed automatically into SMT formulae. Such an automatization is characterized by low manual efforts, which is required for the usage in scenarios of realistic dimensions. Automatization is also required in order to address another aspect of realistic scenarios: the reconfiguration of interaction logic, which may be applied by the later user and thus renders into actual adaptations of the user interface's behavior rather than a simpler change in the physical representation of the interface [24]. Since modern interfaces also cover use cases that make the adaptability of interfaces highly desirable or even necessary, it is obvious that automation is in no way circumventable, should this concept not end up in the proverbial ivory tower. The (repeated) verification of reconfigured user interfaces will be an aspect for future works. In the current work, the focus will be on a transformation-driven approach for validating the reference nets generated from a FILL-based representation by creating an SMT formula from the net. After that, the formula can be combined with the relevant constraints for the test and a solver is able to decide if the interaction logic is correct with respect to the given constraints as previously described above.

In detail, the validation of those colored Petri nets will consist of four steps:

(i) Extract the dynamics of the net.
(ii) Map these information onto a SMT formula.
(iii) Generate the invariants and integrate them into the SMT formula.
(iv) Evaluate the complete SMT formula via SMT solver.

[2]http://www.renew.de

Before giving more detail of this process at hand, the next question to answer is which solver to use in the setup we describe here. We aim for an industry-grade software suite ensuring a high stability together whose further development is also more than likely in the future. For that reasons, we chose Microsoft's Z3[3], introduced in [10], which also offers the great benefit of supporting numerous operating systems (Windows, Linux, MacOS, etc.), processing architectures (e.g. x64, ARM64) and contains bindings for a large part of the currently relevant programming languages (.NET, C/C++, Python, Java, JavaScript/TypeScript/Web Assembly and more).

2.3.1 The Algorithm. A core problem of this transformation is that the logic has to be verified against a formulation of the requirements, which need a similar representation as well in order to generate an evaluable SMT expression. For the first implementations, we will restrict to a simplified variant of Petri nets and consider the restrictions regarding the Java code resulting from the use of reference nets later on, since this first idea is a basic demonstration that the (fully automatic) process is possible in general. Therefore, we assume that the (already known) step of converting a FILL-based representation into a reference net is already performed.

(1) As explained above, all inscriptions which represent the coloring of the reference net are removed.
(2) Afterwards, an initial marking is generated, i.e., only one transition is always active.
(3) From the resulting net, a reachability graph is generated.
(4) The sequences of the firing transitions from this net are extracted:
    (a) A reachability graph is generated.
    (b) From that graph, the path tree is extracted.
    (c) The transition tuples are collected to a fire sequence.
(5) In order to avoid unnecessary complexity, the fire sequences are simplified:
    (a) The sequences are sorted.
    (b) Repeated transitions are deleted.
    (c) Repeated sequences are deleted.
(6) The SMT assertions and declarations are now generated:
    (a) The variable scope is set, as the original variables from the reference net have a very local scope and might collide when used (globally) in an SMT formula. We will simply use an unique variable name by adding the transition ID.
    (b) The SMT assertions are generated.
(7) The SMT assertions for the verifications (e.g. from invariants) are generated.
(8) The SMT formula is generated by adding the assertions in a negated form.

The SMT solver is now started on that formula. If it produces a negative output, it can be concluded that the formula is not satisfiable at all, meaning that there exists no model containing our system together with the opposite of the assertion for the verification. Hence, there exist no errors regarding those criteria.

3 FINAL REMARKS AND FUTURE WORK

In this work, we presented a first insight into a novel approach for evaluating the correctness of existing user interfaces by deriving

[3]https://github.com/Z3Prover/z3

and afterwards validating a formal model in a fully automated way. In order to practically underpin this theoretical work, an implementation is in preparation. Proven and well-supported software suites will be used, but the crucial step of processing will have to be done by custom-built software, as – to our knowledge – this approach has never been considered or even implemented in this way before.

As a sidemark, it should be noted that the computational complexity of this problem is still very high. For example, an "explosion" of the number of states is possible, which should be considered during the implementation. In some cases, reductions of multiple reductions to a single one might help.

The approach presented above provides significant additions to the current situation. However, there are some aspects which would be of major interest in the future. As mentioned earlier, it is of major interest to embed the reconfiguration in our approach.

Another future aspect is the inclusion of Java Code in the network, since for this first approach, we only intend to create a first running demonstrator in order to prove the feasibility of the transformation representations of user interfaces in formal languages into propositional logic formulae.

REFERENCES

[1] T. Allweyer. 2010. *BPMN 2.0: Introduction to the Standard for Business Process Modeling*. Books on Demand. https://books.google.de/books?id=fdlC7K_3dzEC

[2] Eric Barboni, Stéphane Conversy, David Navarre, and Philippe Palanque. 2006. Model-based engineering of widgets, user applications and servers compliant with ARINC 661 specification. In *International Workshop on Design, Specification, and Verification of Interactive Systems*. Springer, 25–38.

[3] Rémi Bastide, David Navarre, and Philippe Palanque. 2003. A tool-supported design framework for safety critical interactive systems. *Interacting with computers* 15, 3 (2003), 309–328.

[4] Bernd Baumgarten. 1996. *Petri-Netze - Grundlagen und Anwendungen (2. Aufl.)*. Spektrum Akademischer Verlag.

[5] Judy Bowen and Steve Reeves. 2008. Formal models for user interface design artefacts. *Innovations in Systems and Software Engineering* 4, 2 (2008), 125–141.

[6] Judy Bowen and Steve Reeves. 2013. Modelling safety properties of interactive medical systems. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*. 91–100.

[7] José C Campos and Michael D Harrison. 2001. Model checking interactor specifications. *Automated Software Engineering* 8, 3 (2001), 275–310.

[8] J Creissac Campos, Michael D Harrison, and Karsten Loer. 2004. Verifying user interface behaviour with model checking. (2004).

[9] Martin Davis, George Logemann, and Donald W. Loveland. 1962. A machine program for theorem-proving. *Commun. ACM* 5, 7 (1962), 394–397. https://doi.org/10.1145/368273.368557

[10] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[11] A Guide. 2001. Project management body of knowledge (pmbok® guide). In *Project Management Institute*, Vol. 11. 7–8.

[12] Roberto J. Bayardo Jr. and Robert Schrag. 1997. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA*, Benjamin Kuipers and Bonnie L. Webber (Eds.). AAAI Press / The MIT Press, 203–208. http://www.aaai.org/Library/AAAI/1997/aaai97-032.php

[13] Olaf Kummer. 2002. *Referenznetze.* Ph.D. Dissertation. University of Hamburg, Germany. https://d-nb.info/965084272

[14] Paolo Masci, Anaheed Ayoub, Paul Curzon, Michael D Harrison, Insup Lee, and Harold Thimbleby. 2013. Verification of interactive software for medical devices: PCA infusion pumps and FDA regulation as an example. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*. 81–90.

[15] Paolo Masci, Yi Zhang, Paul Jones, Paul Curzon, and Harold Thimbleby. 2014. Formal verification of medical device user interfaces using PVS. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 200–214.

[16] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[17] Philippe Palanque, Eric Barboni, Célia Martinie, David Navarre, and Marco Winckler. 2011. A model-based approach for supporting engineering usability evaluation of interaction techniques. In *Proceedings of the 3rd ACM SIGCHI symposium on engineering interactive computing systems*. 21–30.

[18] Philippe A. Palanque, Fabio Paternò, and Bob Fields. 1998. Designing user interfaces for safety critical systems. *ACM SIGCHI Bull.* 30, 4 (1998), 37–39. https://doi.org/10.1145/310307.310359

[19] Carl Adam Petri. 1962. *Kommunikation mit Automaten.* Ph.D. Dissertation. Rheinisch-Westfalisches Institut für Instrumentelle Mathematik an der Universität Bonn.

[20] Jennifer Preece, Helen Sharp, and Yvonne Rogers. 2004. *Interaction design. Oltre l'interazione uomo-macchina.* Apogeo Editore.

[21] João P. Marques Silva and Karem A. Sakallah. 1996. GRASP - a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*, Rob A. Rutenbar and Ralph H. J. M. Otten (Eds.). IEEE Computer Society / ACM, 220–227. https://doi.org/10.1109/ICCAD.1996.569607

[22] Benjamin Weyers. 2012. *Reconfiguration of User Interface Models for Monitoring and Control of Human-Computer Systems.* Ph.D. Dissertation. University of Duisburg-Essen. http://www.dr.hut-verlag.de/978-3-8439-0440-7.html

[23] Benjamin Weyers, Judy Bowen, Alan Dix, and Philippe Palanque. 2015. *Proceedings of the Workshop on Formal Methods in Human Computer Interaction.*

[24] Benjamin Weyers, Dina Burkolter, Wolfram Luther, and Annette Kluge. 2012. Formal modeling and reconfiguration of user interfaces for reduction of errors in failure handling of complex systems. *International Journal of Human-Computer Interaction* 28, 10 (2012), 646–665.

[25] Benjamin Weyers, Barbara Frank, and Annette Kluge. 2020. A Formal Modeling Framework for the Implementation of Gaze Guiding as an Adaptive Computer-Based Job Aid for the Control of Complex Technical Systems. *Int. J. Hum. Comput. Interact.* 36, 8 (2020), 748–776. https://doi.org/10.1080/10447318.2019.1687234

[26] Mihriban Whitmore and Andrea H. Berman. 1996. Common Ground for Critical Shuttle and Space Station User Interfaces: An Independent Verification and Validation Approach. In *Conference on Human Factors in Computing Systems: Common Ground, CHI '96, Vancouver, BC, Canada, April 13-18, 1996, Conference Companion*, Michael J. Tauber (Ed.). ACM, 73–74. https://doi.org/10.1145/257089.257150